4. 事件

English | 中文

服务支持直接调用；事件让插件无需知道有哪些插件正在监听，就能发出通知。harness 使用事件处理工具结果、模型请求和审批决定等交互。

声明、发出与监听

[stats.ts [tmp/cordis-tutorial 中。它是一项负责计数并在每次变化时发出通知的服务：

```
import { Service, type Context } from '@deepseek-ai/cordis'

declare module '@deepseek-ai/cordis' {
  interface Context {
    stats: StatsService
  }
  interface Events {
    'stats/report'(name: string, count: number): void
  }
}

export class StatsService extends Service {
  private counts = new Map<string, number>()

  constructor(ctx: Context) {
    super(ctx, 'stats')
  }

  bump(name: string) {
    const next = (this.counts.get(name) ?? 0) + 1
    this.counts.set(name, next)
    this.ctx.emit('stats/report', name, next)
  }
}
```

```
export const name = 'stats'

export function apply(ctx: Context) {
  ctx.plugin(StatsService)
}
```

`interface Events` 合并与`interface Context` [`ctx.emit`][`ctx.on`[`namespace/action` 命名约定让扁平的事件命名空间保持易读。

[`reporter.ts`:

```
import type { Context } from '@deepseek-ai/cordis'
import type {} from './stats.ts'

export const name = 'reporter'
export const inject = ['stats']

export function apply(ctx: Context) {
  ctx.on('stats/report', (name, count) => {
    console.log(`[stats] ${name} -> ${count}`)
  })
  ctx.stats.bump('tool_call')
  ctx.stats.bump('tool_call')
  ctx.stats.bump('prompt')
}
```

`import type {} from './stats.ts'` 行不会在运行时导入任何内容；它的作用是让 TypeScript 看到声明合并。组合并运行：

```
- name: './stats.ts'
- name: './reporter.ts'
```

```
[stats] tool_call -> 1
[stats] tool_call -> 2
[stats] prompt -> 1
```

[`ctx.on()` 属于 effec`removeListener`绝不需要手动维护

分发模式

`emit` 是 5 种分发模式之一。事件采用哪种模式是其约定的一部分，决定了监听器能否返回值、能否并发运行，以及能否彼此短路：

模式	调用	语义
emit	`ctx.emit(name, ...args)`	同步广播；不会等待或收集返回的 promise 与值。
parallel	`await ctx.parallel(name, ...args)`	所有监听器并发运行，并一同等待。
serial	`await ctx.serial(name, ...args)`	以 `null` / `false` / `undefined` 返回值胜出，并停止后续监听器。
bail	`ctx.bail(name, ...args)`	serial 的同步版本。
waterfall（瀑布式事件）	`ctx.waterfall(name, ...args, next)`	环绕中间件，见下文。

每个 harness 系统都会在其指南中记录其模式。

waterfall：转换或短路

waterfall 将 `next()` 作为最后一个参数传给每个监听器。每个监听器都可以调用 `next()`（将值传递给下游）、跳过 `next()`（短路并返回自己的值），或者在调用 `next()` 之后修改结果。从 `waterfall-demo.ts` 文档把后一种行为称为否决。创建

```
import type { Context } from '@deepseek-ai/cordis'

declare module '@deepseek-ai/cordis' {
  interface Events {
    'demo/transform'(input: string, next: () => Promise<string>): Promise<string>
  }
}

export const name = 'waterfall-demo'

export function apply(ctx: Context) {
  // Listener 1: wrap the downstream result.
  ctx.on('demo/transform', async (input, next) => {
    const downstream = await next()
    return downstream.toUpperCase()
  })

  // Listener 2: short-circuit when it owns the decision.
  ctx.on('demo/transform', async (input, next) => {
    if (input.includes('blocked')) return '** blocked **'
    return next()
  })

  void (async () => {
```

```
    console.log(await ctx.waterfall('demo/transform', 'hello', async () => 'hello'))
    console.log(await ctx.waterfall('demo/transform', 'blocked words', async () => 'blocked words'))
  })()
}
```

[cordis.yml]

```
HELLO
** BLOCKED **
```

[next()] blocked] [next()] [ctx.waterfall] 1

wa next() [next()] waterfall]

harness waterfa agent/request [approval/request]

[cordis.yml]

Image not found or type unknown

#1
3 2026 16:58:44
3 2026 17:02:14